ATLAS METADATA INTERFACES (AMI) AND ATLAS METADATA CATALOGS

Solveig Albrand, Jerôme Fulachier.
LABORATOIRE DE PHYSIQUE SUBATOMIQUE ET DE COSMOLOGIE, IN2P3-CNRS/
Université Joseph Fourier, 53, avenue des Martyrs, 38026 Grenoble Cedex, France.

Abstract

The ATLAS Metadata Interfaces (AMI) project provides a set of generic tools for managing database applications. AMI has a three-tier architecture with a core that supports a connection to any RDBMS using JDBC and SQL. The middle layer assumes that the databases have an AMI compliant self-describing structure. It provides a generic web interface and a generic command line interface. The top layer contains application specific features. Several such applications exist. The principal uses of AMI are the ATLAS data challenge dataset bookkeeping catalogs, and the Tag Collector, a tool for release management.

INTRODUCTION

The "Atlas Metadata Interfaces" (AMI) project [1] started in the spring of 2000 with the requirement to provide an electronic notebook for the Atlas Liquid Argon sub detector test beam acquisition. Other database applications were rapidly requested from the same developers for projects with very similar requirements. In particular, the interface requirements are often almost the same; all projects require an efficient web interface for searching; many projects require a command line interface or a C++ or Java API. Evidently, it makes sense to reuse as much work as possible, and this implies that the architecture of the software must allow development to be generic.

This paper describes the architecture which we have chosen, the functions which the AMI software currently provides, some future plans, and we discuss the main applications. We have now gained sufficient experience to be able to offer some reflections on the advantages and disadvantages of adopting a generic approach.

In the second part of the paper we describe our experience following the introduction of an AMI Web Service in December 2003. This has been globally positive, but some performance problems were encountered.

AMI ARCHITECTURE

Principles

The principles that guided our choice of architecture were:

- A relational database should be used.
- The software should be independent of the particular RDBMS used,
- It should be possible to manage database schema evolution.
- The system should support geographic distribution,
- The interfaces should be as generic as possible,
- The software should not depend on a particular operating system.

It is important to remain independent of a particular relational database, for several reasons. Free databases allow very rapid prototyping. Our initial use of mySQL facilitated the very rapid development of a web interface. However this database lacks, or lacked at the time we began the project, many useful features. It is not certain that free databases such as mySQL or Postgres, are scalable, when compared to the power of Oracle.

Large computing centres that own licenses and have knowledge of, and manpower to support, a database such as Oracle, are unwilling to diversify into providing support for another database. On the other hand, smaller will not be able to use a tool which depends on buying an expensive license and having expert support, and physicists working off-line will not want to install a database server at all, but will need some kind of file management system.

AMI Architecture not only permits different RDBMS to be used, it permits them to be used within the application by the same operation.

Each AMI compliant database contains its own description, in terms of the entities it contains, and their relationships. All our interfaces are designed to exploit these descriptions.

The software is implemented in JAVA, a choice which was motivated by our desire for a multi-platform application, and which has also greatly facilitated the introduction of a web service.

Three Layer Design

Figure 1 shows a schematic view of the architecture of AMI. The lowest software layer is JDBC which handles the database connections. Only three databases are shown in the figure, but currently 6 projects use the AMI base classes. The three main applications are described below.

The middle layer contains two parts. BkkJDBC is a package which wraps the basic SQL requests. It contains light RDBMS specific plugin modules. AMI uses only mySQL at present, the Oracle and Postgres modules are currently under developement . The middle layer also containsm packages that manage the AMI compliant

databases in a generic way, using database descriptions which are within the database itself.. In this layer, no assumptions are made about the names of "databases" ("schema" in Oracle terms), their contained entities or the relations between the entities. Databases can be grouped together, as shown for the Atlas Production Bookkeeping application.

The outer software layer has application specific software. The application specific software in most cases consists of a specialized web page, built on the generic functions. .

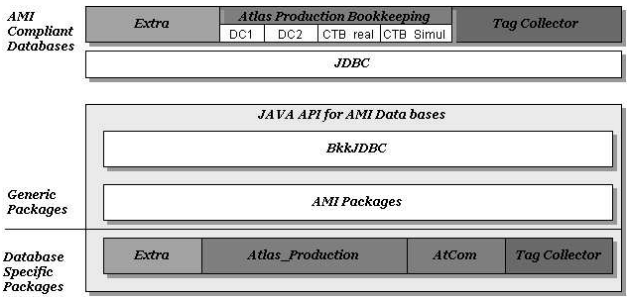

Figure 1: A Schematic View of the Software Architecture of AMI

The AMI deployment model

Figure 2 shows the deployment of the AMI compliant databases on different servers. The client software connects firstly to a router database which is a mechanism for the redirection of client connections to an application database. No AMI software should ever assume that a particular application database will be stored on a particular server. Application databases may be distributed geographically, and may be running with different RDBMS. Client software only needs to be configured for a connection to the router database, and addresses the databases with an application semantic. In this way a database schema can be updated in a transparent way for the user. The client keeps open a connection to the router database, and a number of connections to different databases as required.

AMI Clients

The AMI core software can be used in a client server model as shown in Figure 3. There are 3 possibilities for the client:

- A Web Services client (SOAP).
- From a browser (HTTP) using the AMI web search page.
- By installing the AMI core software on the client side.

Geographic distribution

Geographic distribution is a desirable feature in any tool provided for the Atlas collaboration, which is, of course, itself widely distributed. Our tools should be available at all times in a robust, reliable way. We need to bear in mind that if a tool is to be adopted successfully within Atlas, all features must be able to be scaled up. We have decided to work towards the "data warehousing" model [2] – with several source databases, allowing concurrent input, and central read-only databases updated automatically, which permit complex requests, without affecting the writing efficiency. Such a model has other advantages, for example, individual users may wish to download a database snapshot to their laptop, and may wish to update information whilst working offline. A production site could have complete access rights on a part of the database, and declare that it is ready for uploading to the main database, only when the data has been validated. It should also be pointed out that in this model, different schema could be employed in the source database and the central database. A source database schema should be optimised for the simplicity of input, whereas the database to which queries will be addressed will be optimised for the efficiency of these queries. We are closely following the developments of Grid middleware tools for database distribution, as it would be desirable to use the same mechanisms.

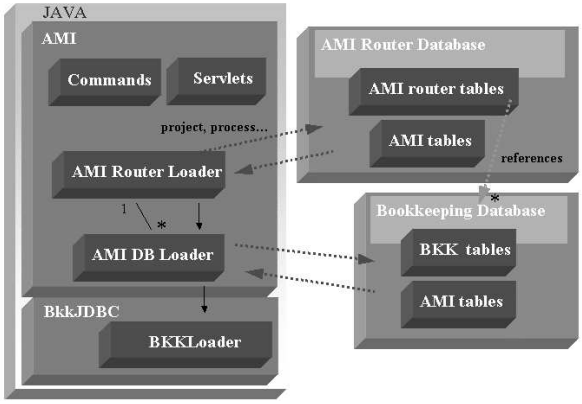

Figure 2 : Redirection of Database Connections

Schema evolution

Schema evolution is a complex database problem to which it is impossible to do justice in this short article. A note describing how AMI software manages various possibilities of schema evolution [3] explains that AMI software is backwardly compatible in terms of the application semantic evolution. It is also to a certain extent forwardly compatible, in that old clients remain largely functional. It turns out that this is often an undesirable feature!

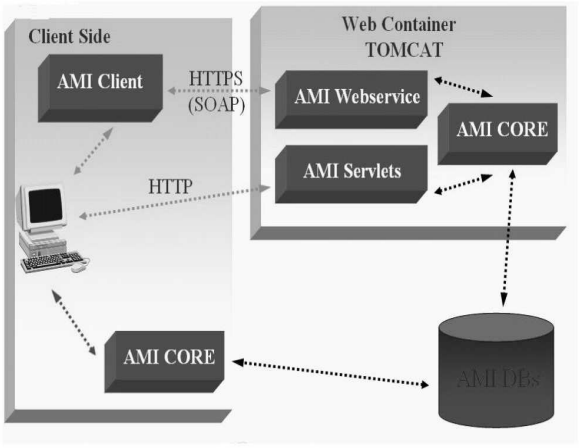

Figure 3 The AMI client possibilities.

GENERIC FUNCTIONS

AMI contains many generic functions which for use either as direct commands, or by servlets.

They include the usual database functions of schema information, input, update and output, and also several functions specific to the web interface, such as the possibility to define a set of actions associated with a row, or a hyperlink associated with a field value.

We are currently working on the implementation of an application level authorization mechanism, which will map users to database roles.

Lastly, we have put in place a task server for the monitoring of coherence, and for heartbeat tests and logging.

THE APPLICATION SPECIFIC LAYER

Atlas Production

The major user of AMI software is the application metadata bookkeeping for the ATLAS data challenges. We have evolved the schema to work with the ATLAS production system for DC2, leaving the DC1 and DC0 data unmoved. Datasets are managed globally using the logical dataset name which must be globally unique. Web interfaces are provided to help in the definition of new datasets. This application has also been used for the 2004 combined test beam runs.

Support for ATLAS distributed analysis has brought new requirements [4], and we are currently refining the design which will meet these requirements with integration of the existing databases.

Tag Collector

Tag collector is a tool designed for assisting in the management of ATLAS software releases. Requirements gathering for Tag Collector II started in mid-2003, and a prototype built on top of the AMI generic layers will be release before the end of November 2004. Tag Collector is be presented in a poster at this conference.

ExTra

ExTra [5] is an AMI based application for the monitoring of external network traffic. It works by interrogating router status at regular intervals – using the AMI task mechanism. Results are archived, and can be displayed on an interactive graphical interface. It is currently installed on various sites within the IN2P3. ExTra has enabled the detection of certain types of worms and viruses present within the monitored sites, and also indelicate or illegal use of the laboratory network infrastructure.

IS THE GENERIC APPROACH USEFUL?

Advantages of the generic database software.

The first and major gain from the generic approach to cataloguing is rapidity in development. Once an AMI compliant database is put in place, no software development whatsoever is needed to provide a complete set of commands and a web search interface. The web interface is totally configurable. For example if the client decides that a combo box should be attached to a parameter shown on the search page, one update to a database table is needed. The addition of a new field requires one database operation, and one AMI Administration command. Thus a rapid prototyping approach towards the correct schema for an application is very easy.

As we have described above, we are able to support applications of a varied nature with little extra software development. Applications level software is able to reuse the generic layers. Indeed the fact that we support several applications has prompted enriching of the generic layers. A good example of this is the current development of Tag Collector II. Many of the features required for this application have a general interest, and so they have been designed as AMI generic software, as part of the Tag Collector project.

Support for schema evolution within AMI is briefly described in section 0. The true database schema is hidden from clients; only the application semantics are exposed. Major schema changes of the application semantic are possible within AMI. Client software is backwardly and, to a certain extent, forwardly compatible. This last feature has lead to some problems

Disadvantages of self-description

Since all AMI commands must read the self-description tables within the database, AMI software accesses the database more times than non-generic software. We were concerned that this could cause performance problems, and this was one of the motivations behind setting up the performance metrics. However we have demonstrated that

any performance loss from this is insignificant when compared with the use of a SOAP based web service.

Use of generic software which aims to support several DBMS implies that we may not be using some of the best, and specific features of each particular database. One might argue that it is wasteful to have Oracle available, and not take advantage of the full set of Oracle features. We believe that by careful design of our database specific plug in modules, and careful factorisation of functionality, we should be able to overcome this objection

Some users of AMI have written their own interfaces to AMI databases, bypassing, or incorrectly exploiting the generic layers of AMI. The resulting software is application specific, and less flexible than AMI's own application specific software. Paradoxically, if the application specific interface becomes popular, we can no longer modify the AMI schema, because it would hinder too many of our colleagues. We welcome specific user tools which are built on top of AMI, and we anticipate that this problem will disappear when an AMI specific management of authorization is introduced which will allow us to block access to the databases using standard mySQL clients. Lastly, we have put a lot of effort into providing a developer's guide and we have introduced a "Wiki" page to in an attempt to improve collaboration mechanisms.[6]

EXPERIENCE WITH WEB SERVICES.

Motivation

In a distributed system, the web service paradigm is ideal as it allows services to be maintained while keeping the interface stable. AMI introduced a web service interface in December 2003. We use the classic SOAP protocol (see Figure 3). Our WSDL [7] is very simple, in fact no specific commands are defined. More specific interfaces can be implemented either as a specific service, or by building a higher-level client.

Available Clients

In theory, any user wishing to use a web service can generate a client for the language and operating system he desires from the WSDL. However, as this is a relatively new technology, client generation seems to be a fairly specialist job, in addition we have found that some SOAP packages used for client generation are incomplete.

The basic AMI web service client is written in JAVA. Other clients have been provided and are advertised publicly as they are become availabl.e [8]

Performance problems

As it is widely known, the SOAP protocol introduces a heavy overhead, and so it become rapidly rather slow. It is almost certainly too slow for use in a distributed analysis environment for HEP. We have put in place a metric to measure AMI performance.

In spite of this poor performance, we have no plans as yet to use alternative implementations. We will follow whatever recommendations are made by the OGSA and EGEE projects. [9] [10]

A CROSS-EXPERIMENT APPROACH TO METADATA.

It is generally acknowledged that dataset selection catalogues containing application metadata are experiment specific, and thus outside the scope of the LCG project. The EGEE middleware group [10] has proposed a common interface for metadata access as part of the gLite package, and we plan to implement this interface for ATLAS distributed analysis.

Even though each experiment has its own ideas and requirements about physics metadata, developers of bookkeeping applications for future experiments have much too learn from more established experiments. A discussion group was recently set up to facilitate contacts between the metadata catalogue developers of several HEP experiments [11] and the AMI team is an active member of this group.

ACKNOWLEDGEMENTS

We have benefited from help and support from many people within ATLAS, in particular from the database group, and from the production system.

Particular thanks are due to:

Julius Hrivnac for producing the C++ Web Service client and Chun Lik Tan (Alvin) for providing the Python client.

REFERENCES

[1] http://atlasbkk1.in2p3.fr:8180/AMI/

[2] The Data Warehouse Lifecycle Toolkit, Ralph Kimball et al, Wiley 1998

[3] http://atlasbkk1.in2p3.fr:8180/AMI/AMI/doc/pdf/ManagingSchemaEvolutioninAMI.pdf

[4] ATLAS Distributed Analysis (These proceedings)

[5] http://lpsc.in2p3.fr/informatique/reseau/extra/

[6] http://atlasbkk1.in2p3.fr:8180/AMI_PUBLIC_WIKI/jsp/Wiki?AMI

[7] http://atlasbkk1.in2p3.fr:8180/AMI/AMI/webService/AMIWebServiceHelp.html

[8] http://www.xmethods.net/

[9] http://www.globus.org/ogsa/

[10] http://egee-jra1.web.cern.ch/egee-jra1/

[11] http://www.gridpp.ac.uk/datamanagement/metadata/